APPLICATION UNDER UNITED STATES PATENT LAWS

Atty. Dkt. No.   PM 273956/TY-50474-US-01
(M#)

Invention: METHOD AND APPARATUS FOR ASSISTING DEVELOPMENT OF PROGRAM FOR VEHICLE

Inventor (s):   ADACHI, Noriyasu

Pillsbury Madison & Sutro LLP
Intellectual Property Group
1100 New York Avenue, NW
Ninth Floor
Washington, DC 20005-3918
                Attorneys
Telephone: (202) 861-3000

This is a:

☐ Provisional Application

☐ Regular Utility Application

☐ Continuing Application
   ☒ The contents of the parent are incorporated
     by reference

☒ PCT National Phase Application

☐ Design Application

☐ Reissue Application

☐ Plant Application

☐ Substitute Specification
   Sub. Spec Filed _____
        in App. No. _____/_____

☐ Marked up Specification re
   Sub. Spec. filed _____
        In App. No _____/_____

SPECIFICATION

DESCRIPTION

METHOD AND APPARATUS FOR ASSISTING

DEVELOPMENT OF PROGRAM FOR VEHICLE

5

TECHNICAL FIELD

The present invention relates to a method and apparatus for assisting development of a vehicle-use program, and in 10 particular to an apparatus and method which improves development efficiency and reliability of a developed program.

BACKGROUND ART

15

Tools for integrated environment utilizing visual programming technique have been developed for facilitating control system design. Computer software for simulation and programming functions is now commonly used to develop vehicle 20 control systems.

When a computer operator working on system development inputs specification data into his computer, using a simulation function, a specification incorporating the concepts input by the user is shown on a computer display in the form of a data

1

or state flowchart. Simulations are then executed according to the charts prepared, and the logic is debugged with reference to the simulation result.

Such simulation software has a function of automatic generation of a program code, such as C language code, based on a chart prepared. C language code prepared in this manner is, however, generally lengthy with many steps and thus not suitable for use intact in vehicle control, which requires severe restrictions on memory and an execution speed. Moreover, such C code is very unfriendly because computer-determined variable names are attached to parts in the charts.

The situation being such, many users choose to manually generate vehicle-use C code suitable for vehicle electric control units (vehicle ECU) while referring to the data and state flowcharts. In particular, dedicated codes having integer logic are generated. These codes differ from general C codes and enable high speed processing using a smaller memory in view of data bit number. This manually generated vehicle-use C code is downloaded to a vehicle-mounted ECU, and debugged.

Conventionally, as described above, the specification simulation process and the program coding process are separate, and each requires a debug operation. Therefore, should a problem be found when debugging vehicle-use code, it is difficult to ascertain whether the problem originated from

2

erroneous coding or an improper original specification. Due to the difficulty in coding and debugging operation, the number of process steps necessary to ensure control program reliability is increased, which in turn prolongs the development period and increases development costs.

The present invention was conceived in light of the above, and aims to provide a method and apparatus for assisting vehicle-use program development to develop highly reliable programs in a shorter period.

DISCLOSURE OF INVENTION

In order to achieve the above objects, according to the present invention, there is provided a method for assisting vehicle program development. In this method, a vehicle control program is generated using a program generator having a function of generating a vehicle-use code from a control specification input. The generated vehicle control program is downloaded to a vehicle ECU, which in turn executes the vehicle control program for debugging the vehicle control program.

As described above, according to the present invention, a vehicle-use code is generated directly from a control specification, and downloaded to a vehicle ECU. Because the original control specification is in common with the

vehicle-use code, bugs are less likely to occur. Therefore, efficient debug operation can be achieved for smaller labor. Moreover, in the person hours required for coding operation can be significantly reduced.

5   Debug operation at the debug step is preferably performed in a program generator which inspects a result of execution of the vehicle control program by the vehicle ECU. Use of a program generator having a control specification can facilitate a debut operation.

10   According to another aspect of the present invention, there is provided an apparatus for assisting vehicle program development, comprising a chart generation function of generating a data flowchart and a state flowchart indicative of a vehicle control specification, and a program code generation function of

15   generating, based on the generated charts, a vehicle-use code for a vehicle control program having an integer logic to be processed by a vehicle ECU. Use of integer logic in the vehicle ECU is preferred because of the limitation of a memory size and an execution speed. In this embodiment, automatic generation

20   of vehicle-use codes are achieved through generation of integer logic codes based on data and state flowcharts.

According to yet another aspect of the present invention, a simulation function is provided for simulating the data flowchart with application of a floating point numbers

4

corresponding to a physical value, and of an integer obtained by converting a floating point number, to output the simulation results. While conventionally a floating point number is handled at a simulation stage, while an integer is handled at

5 a coding stage, according to the present invention, a floating point number and an integer are both handled at a simulation stage, and results of processing with the both are output. With this arrangement, acceptability of integer logic can be easily judged at the specification generation stage. Preferably, a

10 result of back calculation to obtain a floating point number from a result of simulation with an integer applied thereto may be displayed so that a difference between results of simulations with a floating point number applied thereto and of an integer applied thereto, respectively, can be determined.

15 Further, a block symbol in the data flowchart has information on a floating point number, an integer, an integer conversion condition from a floating point number to an integer, and a result of back calculation to obtain an integer from a floating point number using the integer conversion condition.

20 Vehicle-use codes are automatically generated utilizing integer information of the block symbol.

According to still another aspect of the present invention, a priority function is provided for defining an order to execute a plurality of data flowcharts in the same hierarchy in the state

flowchart. With this function, an efficient and appropriate control program can be reliably generated.

According to still another aspect of the present invention, a labeling function is provided to assign a desired label to a selected symbol connection line in a data flowchart, and a vehicle-use code is generated in which the label is used as a variable name for the part to which the label is attached. When the attached label which is easily understandable for a user, a readable vehicle-use code can be generated.

According to yet another aspect of the present invention, a grouping function is provided for grouping, in vehicle-use code generation, a plurality of processes corresponding to a plurality of block symbols in the data flowchart. Preferably, grouping is carried out according to a predetermined grouping restriction condition which defines the number of block symbols to be grouped. Further preferably, a part with a label attached thereto, of a symbol connection line is set as a grouping break. With grouping, readability of a vehicle-use code can be improved.

Vehicle-use code of the present invention can be readily embodied as C language code having been modified so as to suit to a vehicle ECU, but is not limited to C code. A vehicle ECU of the present invention is a desirable ECU to be mounted to a vehicle, such as an engine ECU, a transmission ECU, a

6

suspension control ECU, and a break control ECU. It is needless to say that control specification and vehicle specification which are necessary in program development may be different depending on an ECU and vehicle-mounted devices to be 5 controlled.

BRIEF DESCRIPTION OF DRAWINGS

Fig. 1 is a diagram showing a complete structure of a 10 preferred embodiment of the present invention;

Fig. 2 is a block diagram showing a structure of a program generator;

Fig. 3 is a data flowchart and a state flowchart indicative of control specification;

15 Fig. 4 is a diagram illustrating concept of integer logic for a vehicle ECU;

Fig. 5 is a diagram showing an example of a part of integer logic;

Fig. 6 is a diagram showing structure of an integer block;

20 Fig. 7 is a diagram showing an example of an integer block applied to multiplication;

Fig. 8 is a diagram describing multiplication block processing of Fig. 7;

Fig. 9 is a diagram showing a modified example of the

process of Fig. 8;

Fig. 10 is a diagram showing a support tool for integer logic;

Fig. 11 is a diagram showing a support tool for integer logic;

Fig. 12 is a diagram showing a support tool for integer logic;

Fig. 13 is a diagram showing a support tool for integer logic;

Fig. 14 is a diagram showing a support tool for integer logic;

Fig. 15 is a diagram showing a support tool for integer logic;

Fig. 16 is a diagram showing a support tool for integer logic;

Fig. 17 is a diagram describing a method for determining an operation order;

Fig. 18 is a diagram showing a priority function for defining an operation order in the embodiment;

Fig. 19 is a diagram showing C code resulting from the process of Figs. 17 and 18;

Fig. 20 is a diagram showing example C code generated based on a data flowchart;

Fig. 21 is a diagram showing a C code generated based on

8

the same data flowchart as that in Fig. 20, and given labeling in the embodiment;

Fig. 22 is a diagram showing grouping process in C code generation;

Fig. 23 is a diagram showing variable name data base and expression database which are generated in C code generation;

Fig. 24 is a diagram showing grouping process utilizing the database of Fig. 23;

Fig. 25 is a diagram showing an example of grouping process; and

Fig. 26 is a diagram showing an example of a display screen when the program generator monitors engine control.

BEST MODE FOR CARRYING OUT THE INVENTION

In the following, a preferred embodiment of the present invention will be described with reference to the accompanying drawings.

Referring to Fig. 1, a program generator 1 has functions of generating and simulating a control specification, and for generating vehicle-use C code from the generated control specification.

The program generator 1 comprises a computer, having, as shown in Fig. 2, a CPU 10, a ROM 11, a RAM 12, a keyboard 13,

9

a pointing device 14, a display 15, a hard disk 16, a communication circuit 17, and a CD-ROM drive 18. A program for achieving the respective functions of the program generator 1 is stored in either the ROM 11, the hard disk 16, or the CD-ROM

5   19, and executed by the CPU 10. An input/output device is not limited to a keyboard 13, a pointing device 14, and a display 15. Any type of memory device other than a CD-ROM 19, such as DVD, may be used. Any type of memory device other than a hard disk 16 may be used.

10   The user writes a specification incorporating his idea on the display 15 by operating the keyboard 13 and the pointing device 14 (a mouse and so on), and thereupon the control specification is shown on the display 15 in the form of a data flowchart (a block diagram) and a state flowchart (a state

15   transition chart, a state transition diagram), as shown in Figs. 3(a), 3(b), a data flowchart showing a partial data flow, a state flowchart showing an entire control flow.

The user also inputs a vehicle specification by operating the keyboard 13 and the pointing device 14 so that a hypothetical

20   vehicle model is formed in the program generator 1. Specifically, a vehicle model is formed as a collection of expressions which describe various vehicle motions and operations, and so on.

In response to the user's instructions, the generated

specification (data and state flowcharts) is simulated. Specifically, as shown in Fig. 1, the vehicle model is controlled on the computer according to the control specification logic. When a simulation result is shown on the

5 display 15, necessary debugging is applied. In a debug operation, the specification is modified and improved for completion.

When the specification completes, vehicle-use C code is generated based on the complete specification in response to

10 the user's instruction. Specifically, a file containing code written in the C language and corresponding to the data and state flowcharts is formed. A vehicle-use C code has an integer logic and a structure enabling higher speed processing using a smaller memory in size, compared to general C code (described

15 below). A resultant vehicle-use C code is downloaded to an actual vehicle ECU 2, e.g., an engine ECU.

The vehicle model is input from the program generator 1 to the vehicle model device 3. The vehicle model device 3, including a DSP, simulates high speed interruption and other

20 characteristics of the vehicle ECU 2. Conducting such a simulation before testing using an actual vehicle reduces development period and results in a more effective product. After the vehicle model device 3 is connected to the vehicle ECU 2, the vehicle-use C code, having been loaded into the ECU

11

2, is executed to begin simulation in a near-real environment. Operation of the vehicle ECU 2 and vehicle model device 3 are monitored by the program generator 1 via the communication circuit 17, and a debug operation is applied in the program generator 1. For example, should improper control operation be found, the cause thereof is detected using the data and state flowcharts. Then, improper points are amended in the computer, and the amended logic is verified.

After debugging is completed, the vehicle ECU 2 is mounted to a vehicle 4 for actual testing. In actual testing, any defect in need of amendment due to a non-linearity factor and so on of an actual vehicle is detected for final program amendment.

In the following, a program generator 1 will be described in more detail.

Control Specification Input/Generation Function (Integer Logic)

Conventionally, at a control specification input stage, a chart is prepared for processing a floating point number indicative intact of a physical value. Next, a simulation is carried out using the floating point number, so that acceptability of the control specification can be determined based on the simulation result. Therefore, C code which is automatically generated from the chart using a conventional

function contains a floating point number.

Here, vehicle control requires making best use of the capacity of the CPU of an ECU in view of minimizing costs and increasing processing speed. This demand is particularly significant in control of devices with high speed rotation, such as an engine. In order to meet such a demand, preferably, integer logic (a fixed point number) is applied to C code for a vehicle-use ECU to generate C code with a data bit of a different number from that of general C code. This is a significant difference between vehicle-use C code and automatically generated general C code, and a major reason for forcing manual generation of vehicle-use C code.

In this embodiment, an integer block (described later) is introduced so that integer logic can be taken into consideration as early as at a control specification input stage, and whereby vehicle-use C codes having an integer logic can be automatically generated from a control specification.

(1) Conversion from Floating Point Number to Integer

A floating point number is converted into an integer, following the below expression (1).

$$x\_int=int(x\_float-OFFSET)/SLOPE \quad \ldots(1)$$

13

wherein x_float is a floating point number, x_int is an integer, OFFSET (or bias) and SLOPE (or LSB) are integer conversion condition.

5   (2) Summary of Integer Logic

Fig. 4 illustrates the concept of integer logic in a vehicle ECU. The sensor 21 sends a voltage signal v according to a detected physical quantity p to an A/D converter 22. The A/D converter 22 converts the received voltage signal v into
10   integer data P to send to the vehicle ECU 2. Using integer logic, the vehicle ECU 2 obtains integer data Q for a control parameter from the integer data P of a detection signal. Integer data may be, e.g., an unsigned integer of 16 bits. The integer data Q is converted into a control signal to be output to an actuator
15   23, or an object to be controlled, for generation of physical quantity q.

In Fig. 4, the integer logic must produce a result equivalent to that which would be produced according to the original control logic with respect to a physical value. SLOPE,
20   OFFSET at the input/output portion of the vehicle ECU 2 are lp, lq, op, oq, respectively. In actuality, integer logic includes a plurality of operation process stages, and has many SLOPEs, OFFSETs in an intermediate process. These SLOPEs, OPPSETs should be set such that the integer logic is optimized in view

14

of a memory size and an execution time.

In a case wherein a control specification (logic) generated by a user includes a transfer function, as shown in Fig. 5, and identical SLOPE is used for all seven items of the transfer function in constituting an integer logic, the seventh item may be always caused zero when SLOPE which is suitable for the first item is applied to all seven items because factors differ significantly between the first and seventh items.

Generally, too small a SLOPE may cause overflow, while too large a SLOPE may result in negligence of an input of a sensor. Further, setting an improper SLOPE may result in a control logic containing a lengthy component. In order to avoid these problems, readily setting of a suitable integer conversion condition is desired.

(3)   Integer Block

In order to generate a suitable integer logic in a data flowchart so that suitable integer conversion condition can be readily set, an integer block, shown in Fig. 6, is introduced in this embodiment. An integer block is a block symbol used in a data flowchart, which a user can write into a data flowchart.

As shown in Fig. 6, an integer block includes six types of information pieces, namely (1) integer, (2) SLOPE (LSB), (3) OFFSET (bias), (4) float (a floating point value), (5) Data Type,

(6) "integer * SLOPE + OFFSET". (6) is a back float, or a result of back calculation to obtain a float from an integer.

Function of an integer block will be described referring to Figs. 7 and 8 and using multiplication as an example.

5   In the data flowchart of Fig. 7, input data 1, 2, and SLOPE and OFFSET for each data are input to a multiplication block. The multiplication block obtains the above mentioned six types of data items from these inputs, and outputs them.

Fig. 8 shows an expression describing a process in a

10   multiplication block. In this process, input values x, y are converted into integers using offsets 1, 2, and slopes 1, 2, respectively, and a product of the two integers is output (integer output). Meanwhile, a float output is "xy". Comparison between expression (1) and "xy" leads to SLOPE and

15   OFFSET on the output side (Fig. 8, expressions (2), (3)). Further, a back float (=integer * $SLOPE_{OUT}$ + $OFFSET_{OUT}$) is also output.

The user can ascertain the simulation result by referring to the display in Fig. 7. "Float" indicates whether or not basic

20   control logic is correct. Further, "float" and "back float" are compared. When the difference (0.1) is within a tolerable range, it is known that integer logic is appropriate. On the other hand, when the difference is not within the tolerable range, it is known that inappropriate integer logic, or integer

conversion condition, namely "slopes 1, 2", is set. Then, "slopes 1, 2" are repeatedly adjusted until an appropriate result is obtained.

Figs. 7 and 8 show only a part of a control specification. In actuality, control specification constitutes of a series of numerous connected blocks. In this embodiment, "SLOPE" can be verified and adjusted in each block in a control process. For example, when a significant difference is found between "float" and "integer" in a certain block, integer conversion condition is adjusted in any block preceding the focused block.

With the above arrangement, "float" and "integer" can have substantially equivalent meaning over the entire control process. In other words, a situation with a difference between "float" and "integer" becoming larger as it goes to latter blocks, can be avoided. Also, a situation in which input of a certain sensor is ignored because of an improper SLOPE set, can be avoided.

As shown in the third expression in Fig. 8, OFFSET on an output side includes input values x, y. When the influence thereof is not negligible, the user can chose modified process, as shown in Fig. 9. Specifically, "offset/slope" is added to an integer prior to the multiplication whereby the item for "offset" on the output side becomes zero, as is shown in the first expression in Fig. 9.

17

Note that, although multiplication is used as an example in the above to describe a function of an integer block, similar function can be set to other operations. Figs. 10 to 16 show various integer blocks which support integer logic. Although

5   information y(5) to y(8) are added to an integer block in this tool, as shown in Fig. 10, basic information y(1) to y(4) is the same as described above. Also, LSB in Figs. 10 to 16 corresponds to "slope".

As described above, in this embodiment, as a result of

10   introduction of an integer block, a control specification in the form of a chart expresses an integer logic. As suitable integer conversion condition can be set through comparison between theoretical control logic (real number operation) and integer logic (integer operation), suitable integer logic

15   including fewer bugs can be easily made. The complete control specification itself is usable for a computer simulation. In addition, vehicle-use C code can be automatically generated by extracting an integer logic part from the specification. As the control specification and vehicle-use C code have parts in

20   common, bug generation is reduced at a code generation stage, and program verification, such as debugging, can be easily applied.

Priority Function (Regulation of Operation Order)

18

For determination of an order to execute a plurality of modules or operational expressions, generally, each module or operation expression is made into a triggered subsystem and given a function call from a state flowchart, as shown in Fig. 17.

5

In this embodiment, a priority function is additionally provided, as shown in Fig.18. Specifically, the user imparts a number indicative of a priority order to the data flowchart, whereby each module or operation expression is made as a prioritized subsystem (1_Prior_Subsystem, 2_Prior_Subsystem). Operations are executed in the order of names of the subsystems.

10

The priority function leads to an advantage that an order to execute operations in data flowcharts in the same hierarchy can be easily determined. In addition, another advantage can be achieved that a C cod which is easier than conventional one can be generated. This is preferable in view of readability and a memory capacity.

15

C Code Generation (Labeling)

20

C code generation is applied in response to a user's instruction. As described above, as a result of application of an integer block, each block in a data flowchart has integer information. Therefore, a series of block groups can be processed using integer operations. By generating integer

19

operations, vehicle-use C code having an integer logic are generated. Specifically, referring to the example in Fig. 7, integer operations (two integer inputs and one integer output) are extracted, at a chart level, from information groups to be

5   input or output with respect to a multiplication block.

In order to modify an extent software product for the above use, a tool for automatically producing a general C code based on a chart and a modeling tool into which a C code is written are prepared, and the model is modified so as to suit to the

10   integer logic. Then, processing which is absent from a general C code and unique to a dedicated command, and a dedicated command resulting from a modified general C code are also applied.

Fig. 20 shows an example of a C code generated from a data flowchart. According to a general C code generation method,

15   computer-determined variable names are given to the respective parts, as shown in the drawing. This results in a difficult to read C code unfriendly to human users.

That is, readability is not fully taken into consideration in a general C code automatic generation method,

20   which is mainly concerned with coincidence between a C code and a chart. Nevertheless, often a program developer must read a C code in testing using an actual ECU, which is conducted for motion verification in vehicle system development. Therefore, user friendliness is desired in automatic vehicle-use C code

generation.

In this embodiment, a labeling function, as shown in Fig. 21, is employed to improve readability. A user clicks a connection line in a block symbol in the data flowchart, in response to which a desired label is attached to the selected connection line. In Fig. 21, as an example, labels "t_x", "t_y", "t_z", "t_xyz" are attached. In actuality, more easily understandable labels, such as a module name, a sensor name, and so on, are preferably attached. In C code generation, the label may be used as a variable name of the part with that label attached. As a result, easily readable C codes are generated, as shown in Fig. 21.

Note that labels may be attached to only a desired line, rather than to all connection lines, as shown in Fig. 21, as the former may help easy reading of C codes. As for a line without labeling, a computer-determined variable names may be used intact.

Grouping in C Code Generation

(1)   In general C code generation, a different variable name is given to each block in a data flowchart, and an expression for introducing one variable is generated corresponding to each block.   Therefore, resulting C code contains numerous variables and expressions, making it long and less readable.

In order to address this problem, in this embodiment, a grouping function is employed to improve readability. In Fig. 22, there are two blocks intervening between input and output. With C code on the left side without grouping, two expressions are generated according to the number of blocks. With C code on the right side with grouping, operations for two blocks are integrated into a single expression, in which variable names unnecessary for a user reading a C code are deleted, and easily readable C code is thereby generated.

Here, grouping of too large a number of blocks may result in a lengthy expression with poor readability. To address this problem, a condition for defining the number of blocks contained in one group is preferably determined. In this example, the maximum number of such blocks is set at two. That is, for one expression of a C code, operations will be carried out twice at maximum.

Grouping process will next be described with reference to Figs. 23 and 24. In C code generation, ID is attached to each block in a data flowchart for use as a variable name. Then, a variable name (signal name) database is made for correlating ID and a user-designated label (Fig. 23 (a)). When no corresponding label can be found, ID is used intact.

In Fig. 24, ID of an input signal to a focused block is detected, using a function to detect an input signal (S11).

Then, whether or not grouping condition (twice at maximum) is held with each input signal is determined with reference to an expression database (Fig. 23 (b)) (S12) and when the result is negative (less than twice), the expression database is searched

5   (S13) to adopt an expression corresponding to an input ID. On the other hand, when it is determined that the condition is held, a variable name database is then searched (S14) to employ a corresponding label as a variable name. When no attached label is detected, an ID is used.

10   Using an expression employed at S13 or a variable name employed at S14, an expression for the focused block is generated (S15). Whether or not a grouping condition is held with respect to the generated expression is detected (S16). When it is not, the expression generated at S15 is registered

15   to the expression database for preparation for the process in the next block (S17). On the other hand, when the condition is held, the expression generated at S15 is written into a file (S18). In Fig. 22, an expression for two blocks combined is output.

20   With reference to Fig. 25, a specific example of the above grouping processing will be described

With input signals S1, S2, an expression database is searched with respect to an input signal S1 to obtain an expression (x+1). As the number of grouping for this expression

is only once (i.e., the above mentioned condition is held), the expression (x+1) is used as an input signal. Then, a search is carried out with respect to an input signal S2. As the number of grouping for an input signal S2 is twice, a variable name database is then searched to adopt label x2 as an input signal. Therefore, an operation within the focused block will be "(x+1)*x2" (multiplication).

Next, an output signal t_x is retrieved from the variable name database. As two groupings have been conducted with respect to the input signal S1, the grouping condition is held. Then, t_x=(x+1)*x2 is output. If grouping condition is still to be held at this stage, the generated expression is registered in the database.

(2)   In grouping processing, a part in a chart, where a label is attached is always determined as a grouping break. All labels attached by the user are contained in an operation expression for a C code. When labels attached to an appropriate part are assigned from a chart prepared by a human user, C code having a structure corresponding to that of the chart is generated. With this processing, more easily readable C code is automatically generated.

In an example wherein a user attaches a label to the beginning and end of a data flowchart, very readable C code is

24

automatically generated, which has definite variables at the beginning and end thereof and process segments in appropriate intervals.

5　　As described above, according to this embodiment, when grouping is applied in C code generation, readability of an automatically generated C code can be improved, and memory can be conserved through reduction of variable names.

Download to Vehicle ECU

10　　As described above with reference to Fig. 1, the program generator 1 downloads an automatically generated vehicle-use C codes to the vehicle ECU 2 in response to a user's instruction. The vehicle ECU 2 then executes the downloaded vehicle-use C code to control the vehicle model device 3 (DSP). The program

15　generator 1, monitoring the control being executed, displays the execution state in an appropriate format on the display. Fig. 26 shows an example of a monitor screen when the vehicle ECU 2 is an engine ECU. The user operates the program generator to conduct motion verification and a debug operation with

20　respect to a vehicle-use C code.

In this embodiment, a suitable vehicle-use C code having few bugs is generated. The vehicle-use C code is in common with the original control logic. The vehicle-use C code has higher readability. Therefore, a debug operation can be easily and

25

efficiently performed.

As described above, according to the present invention, as a vehicle-use code corresponding to the control specification input by the user is automatically generated, the number of coding operation steps can be significantly reduced. The input control specification itself is usable for computer simulation, as well as usable in a vehicle-ECU after conversion into a vehicle-use code. This enables significant reduction of the number of debug operation steps . As a result, the number of steps necessary to ensure reliability can be reduced, as well as a development period and costs. The advantage of the present invention for assisting a process from preparation of a specification to logic debugging, is remarkable particularly in a situation with overgrown and complicated vehicle control programs.

Also, according to the present invention, a highly readable vehicle-use code can be automatically generated. As a result, the number of steps for a debug operation and so on can be further reduced.

INDUSTRIAL APPLICABILITY

As described above, a method and apparatus for assisting development of a vehicle-use program according to the present

invention can be utilized in development of a vehicle control

program to be executed by a vehicle ECU.